**الگوی Bridge یا پل**

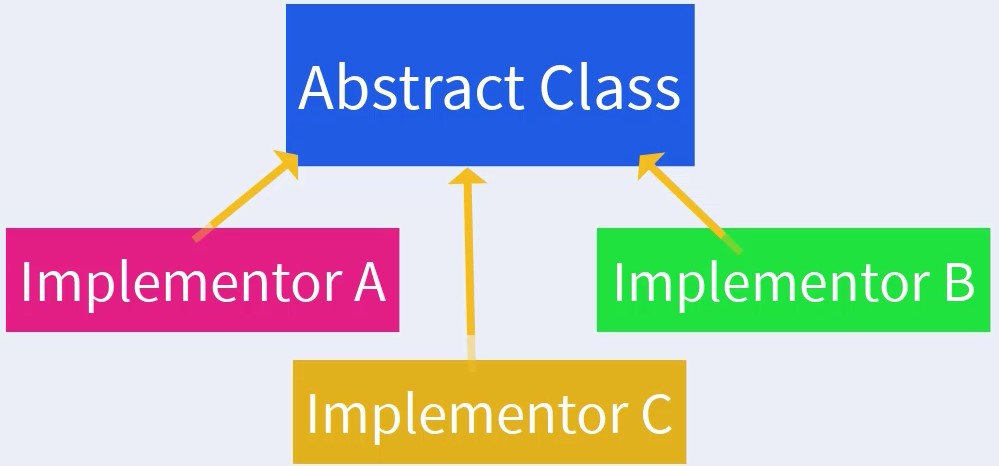
**هدف پیاده سازی:**

**جداسازی انتزاع و پیاده سازی**

**یادآوری: اگر بخواهیم برای یک کلاس چندین پیاده سازی را ارائه دهیم:**

**ایجاد یک کلاس Abstract و به ازای این کلاس چندین پباده سازی میتوانیم ایجاد کنیم که این کلاس ها همگی از abstract ارث بری میکنند.**

**مشکل این روش این است که همه کلاس های پیاده سازی شده اتصال محکمی به کلاس abstract ما دارند ( یعنی هر متدی به کلاس abstract اضافه شوند ،کلاس های پیاده سازی باید آن متد را پیاده سازی کنند.هرکدام آن را پیاده سازی نکنند برنامه compile نمی شود و نمی توان abstract را مستقل از پیاده سازی ها توسعه داد.)**

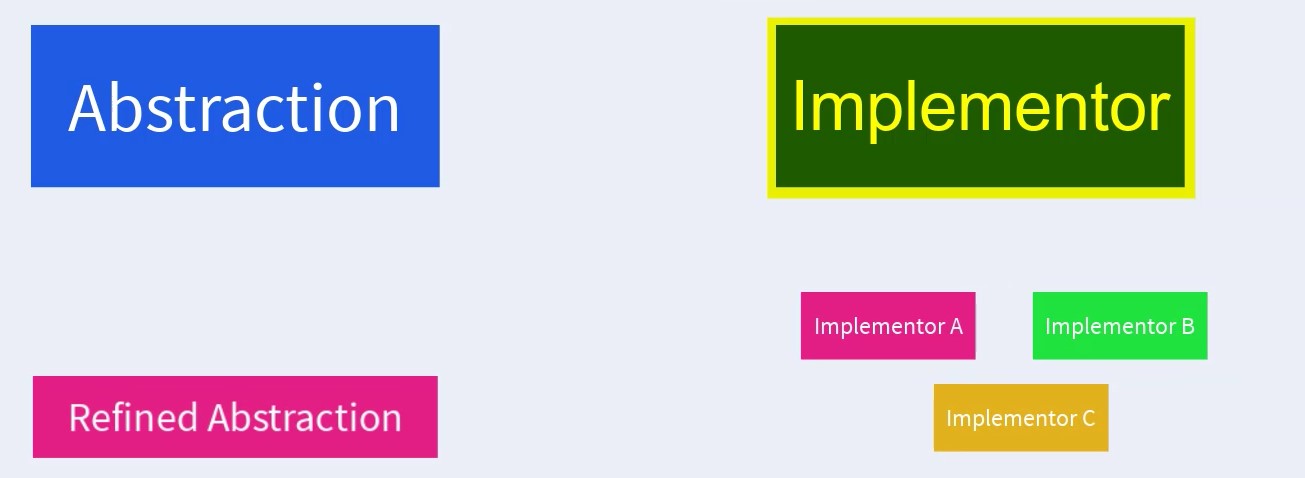
****

**حال الگوی Bridge کلاس abstract را به دو دسته ی abstraction و implementor تقسیم میکنند.(هر دو کلاس abstract هستند)**

کلاس **abstraction برای نمای بیرونی کلاس همان متد ها و property ها ( client می بیند.)**

**کلاس Implementor یا bridge برای پیاده سازی های داخلی کلاس(پیاده سازیabstraction )**

**و از کلاس Implementor یا bridge می توانیم چندین پیاده سازی داشته باشیم.**



Refiend Abstraction کلاسی ارث بری شده از abstraction است که به دلیل اینکه نمیتوانیم از کلاس abstraction نمونه ایجاد کرد از این کلاس استفاده می شود.

نکات این الگو

1. اگر یک Implementor و ConcreteImplementor داریم می توانیم آن ها را با هم یکی کنیم.

برای نمونه در مثالی که نوشته شد اگرما تنها از GmailService برای ارسال ایمیل استفاده میکنیم دیگر نیازی به ساخت و استفاده از IMailServiceBridge نبود.(حالت ایده آل این الگو نیست و بهتر است همانند مثال انجام داد)

1. در حالت عادی در مثالی که نوشته شد client های ما میتوانند به صورت مستقیم از ConcreteImplementor ها استفاده کنند ( یعنی می تواند بصورت مستقیم از GmailService یا YahooService نمونه سازی کرده و از آن استفاده کند) برای این کار می توانیم این سرویس ها را در یک assembly دیگری قرار دهیم.(اضافه شدن ClassLibrary با نام MailServiceImplementor )
2. این الگو شباهت بسیار زیادی با الگوی strategy دارد.
3. متد ها ی Abstraction و implementor بایدباهم تفاوت داشته باشند.
4. میتوانیم به ازای یک Abstraction ، چندین Implementor داشته باشیم.
5. این الگو به تنهای استفاده نمی شود.در کنار این الگو از تکنیک های تزریق وابستگی و IOC هم استفاده میکنیم.
6. Abstraction شباهت زیادی به الگوی Facade دارد.
7. این الگو شباهت بسیاز زیادی به تکنیک تزریق وابستگی دارد ،اگر در برنامه خود از تزریق وابستگی استفاده میکنید،دیگر نیازی به استفاده از الگوی Bridge نیست و بهتر است از تزریق وابستگی استفاده شود.( به دلیل انعطافی که تزریق وابستگی دارد)

مزایا

1. برقرای اتصال سست
2. اجبار به رعایت اصل OCP
3. اجبار به رعایت اصل SRP
4. اجبار به رعایت اصل SOC
5. Abstraction و Implementor را می تواند به صورت مستقل از هم توسعه داد.
6. از کامپایل های اضافی جلوگیری میکند.
7. لایه بندی کد های برنامه

معایب:

1. اگر واسط تغییر کند همه چیز خراب می شود؛

برای مثال اگر ما یکی از متد های IMailServiceBridge راتغییر بدهیم،باید MailServiceAbstraction را هم تغییر دهیم.

چه زمانی می توان از این الگو استفاده کرد:

1. زمانی که میخواهیم پیاده سازی ها را در زمان اجرا جایگزین کنیم(بدون داشتن کامپایل یا بدون تغییر کد)
2. انتزاع و پیاده سازی را مستقل از هم توسعه دهیم.
3. بخواهیم پیاده سازی ها را تغییر بدهیم بدون نیاز به تغییر انتزاع و کلاینت
4. توسعه پیاده سازی ها نیاز به کامپایل کلاینت نداشته باشد.(فقط کامپایل dll مربوط به این پیاده سازی ها)
5. زمانی که بخوایهم انتزاع و پیاده سازی به هم اتصال محکم نداشته باشد( ارث بری نشود)
6. قصد داریم علاوه بر پیاده سازی های گذشته ،پیاده سازی های جدیدی را هم اضافه کنیم.(افزودن پیاده سازی با قابلیت نگه داری پیاده سازی های قبلی)